Tietojenkäsittelytiede **24** Joulukuu 2005 sivut 8–21
Toimittaja: Jorma Tarhio

Historialaajennus mallipohjaisessa testauksessa

Timo Kellomäki
Tampereen teknillinen yliopisto
Ohjelmistotekniikan laitos

1 Johdanto

Testauksen tavoite on kerätä luottamusta siihen, että testattava järjestelmä toimii oikein. Mitään todellista järjestelmää ei voida testata läpikotaisin, joten on olennaista suunnitella testaus niin, että järjestelmän käyttäytymisestä tutkitaan toisaalta mahdollisimman paljon ja toisaalta mahdollisimman olennaisia osia. Rinnakkaisissa järjestelmissä testaus on tavallisiakin järjestelmiä hankalampaa, koska rinnakkaisuus aiheuttaa epädeterminismiä, eli järjestelmä ei välttämättä käyttäydy samoilla syötteillä aina samalla tavalla. Näin järjestelmässä voi piillä virheitä, jotka saadaan esiin vain hyvin harvoin.

Mallipohjaisessa testauksessa järjestelmän toimintaa mallinnetaan formaalilla, esimerkiksi tilakonemuotoisella, spesifikaatiolla. Spesifikaation määrittelemän käyttäytymisen kattava joukko testitapauksia on mahdollista luoda automaattisesti, mutta se ei tietenkään tarkoita, että itse testattava järjestelmä olisi katettu. [5]

Jos järjestelmän spesifikaatio on korkealla abstraktiotasolla, saattaa spesifikaation täysin kattava testijoukko olla hyvinkin pieni. Tällöin olisi tarpeen laajentaa spesifikaatiota niin, että sen määrittelemä käyttäytyminen ei muutu, mutta sen perusteella luotavat testitapaukset kattavat toteutuksen mahdollisimman monipuolisesti.

Tätä tilannetta varten on kehitetty historialaajennus, joka on automaattinen tapa laajentaa spesifikaatiota siten, että mallipohjaisen testauksen menetelmät luovat laajennetun spesifikaation pohjalta testitapausten joukon, joka syventää testikattavuutta hyvin luonnollisella tavalla.

Tässä artikkelissa esitellään historialaajennuksen toimintaperiaatteet sekä pintapuolisesti sen ymmärtämiseen vaadittavia taustoja, ja lopuksi tutkitaan sen tehokkuutta virheiden löytämisessä. Luku 2 esittelee menetelmän teoreettisena pohjana olevat tilakoneet. Luku 3 käsittelee samoilu- ja mallipohjaista testausta, joiden yhteydessä historialaajennusta voidaan soveltaa. Luku 4 käy läpi itse historialaajennuksen idean ja pohtii sen toteuttamista teoriassa ja käytännössä. Luvussa 5 vertaillaan historialaajennuksen tehokkuutta perinteisiin menetelmiin eräillä esimerkkijärjestelmillä.

2 Tilakoneet

Järjestelmien toimintaa voidaan kuvata formaalisti *tilakoneilla* [7], jotka ovat olennaisesti suunnattuja graafeja. Solmut eli *tilat* vastaavat järjestelmän tiloja ja kaaret eli *tilasiirtymät* erilaisia tapahtumia, jotka muuttavat tilaa. Kullakin tilasiirtymällä on myös nimi, joka kertoo, mitä tapahtumaa kaari esittää. Tilasiirtymien

nimet voivat olla hyvin konkreettisia järjestelmän tapahtumia tai abstraktimpia tapahtumakokonaisuuksia. Näin sama järjestelmä voidaan kuvata monella eri abstraktiotasolla. Tilakoneen *aakkosto* sisältää kaikki siinä esiintyvien tilasiirtymien nimet ja mahdollisesti muitakin tilasiirtymien nimiä (tällä on merkitystä synkronisessa rinnankytkennässä, josta puhutaan pian). Tässä esityksessä tilakoneen *koko* tarkoittaa sen tilojen määrää.

Tilakoneen *suoritus* alkaa jostakin *alkutilasta* ja etenee kulkemalla pitkin jotakin tämänhetkisestä tilasta lähtevää kaarta eli suorittamalla *vireessä oleva tilasiirtymä*. Näin suoritus voi jatkua teoriassa loputtomiin. Toisin kuin äärellisissä automaateissa, erillisiä lopputiloja ei tilakoneissa ole. Esimerkkejä tilakoneista on nähtävissä myöhemmin artikkelissa, katso esimerkiksi kuva 3.

Useat erilaisilta näyttävät saman abstraktiotason tilakoneet voivat esittää samaa käyttäytymistä hyvin paljon samaan tapaan, kuin sama algoritmi voidaan kuvata monella aivan erilaisella C-kielisellä ohjelmalla. On olemassa algoritmeja, joilla voidaan minimoida annetun tilakoneen koko muuttamatta sen käyttäytymistä.

Tilakoneet soveltuvat erityisen hyvin rinnakkaisten järjestelmien kuvaamiseen, sillä monimutkaisen järjestelmän eri osat voidaan mallintaa pieninä ja yksinkertaisina tilakoneina, jotka voidaan sitten asettaa vuorovaikuttamaan toistensa kanssa esimerkiksi jaetun muistin tai sanomajonojen välityksellä. Tässä esityksessä käytetään yleistä ja abstraktia vuorovaikutusmenetelmää, *synkronista rinnankytkentää*. Sen avulla voidaan mallintaa muun muassa edellä mainitut kommunikaatiomenetelmät.

Synkronisen rinnankytkennän kannalta osa tapahtumista on *näkyviä*. Kaikki ne rinnankytkentään osallistuvat tilakoneet, joiden aakkostoon tietty näkyvä tapahtuma kuuluu, toimivat tämän tapahtuman suhteen synkronisesti. Tällainen tapahtuma voidaan suorittaa vain, jos se on vireessä kaikissa näissä tilakoneissa, ja tällöin sen täytyy tapahtua kaikissa niissä yhtäaikaisesti. Näin yksikin sellainen rinnankytkentään osallistuva tilakone, jossa tapahtuma ei ole vireessä mutta jonka aakkostoon tapahtuma kuuluu, estää kaikkia muita suorittamasta sitä. Loput tilasiirtymät ovat *näkymättömiä*, jolloin tilakoneet voivat suorittaa niitä täysin riippumatta toisistaan.

Tällaisen rinnankytkennän lopputulos on yhä tilakone, jonka tilat vastaavat siihen osallistuneiden tilakoneiden yhteistiloja, eli se saadaan muodostamalla karteesinen tulo rinnankytkettävien tilakoneiden tiloista (ja poistamalla tilat, joita ei voida saavuttaa). Tulos siis kuvaa rinnankytkettyjen tilakoneiden yhteistä käyttäytymistä.

3 Mallipohjainen testaus ja samoilutestaus

Testitapaus tarkoittaa jotakin tyypillisesti lineaarista syötteiden ja niihin odotettujen tulosteiden sarjaa, jonka tehtävä on yleensä testata jotakin yksittäistä asiaa. *Testiajolla* taas tarkoitetaan järjestelmän käynnistämisestä alkavaa ja pysäyttämiseen loppuvaa testaustapahtumaa, joka saattaa sisältää useitakin perättäisiä testitapauksia.

Tilasiirtymät jaetaan usein testauksen yhteydessä *syötteisiin* (input) ja *tulosteisiin* (output). Jokainen tilakoneen näkyvä tilasiirtymä on tällöin joko syöte tai tuloste. Sama tilasiirtymä voi olla toisen tilakoneen syöte ja toisen tuloste. Nyt kahden tilakoneen rinnankytkennässä synkronisesti suoritettavat tilasiirtymät ovat epä-

symmetrisiä syöte/tuloste-pareja: kun toinen tilakone antaa jotakin ulospäin, toinen ottaa sen vastaan omana syötteenään. Usein syötteet merkitään kysymysmerkillä ja tulosteet huutomerkillä.

Järjestelmää testataan antamalla sille syötteitä, jotka ovat siis testaajan näkökulmasta tulosteita. Takaisin saadaan järjestelmän tulosteita, jotka ovat testaajan syötteitä. Yleisessä tapauksessa järjestelmän ei tarvitse perustua syötteen ja tulosteen vuorottelulle, vaan useita syötteitä voidaan antaa peräkkäin ja samassa tilassa voi olla yhtä aikaa sallittua antaa lisää syötteitä tai kuunnella järjestelmän vastetta. Jos järjestelmä ei vastaa kuunneltaessa jossakin tilassa millään tulosteella, sen katsotaan antaneen erityisen tulosteen, *hiljaisuuden*, jota merkitään δ:lla.

Spesifikaatiossa δ voi olla tilasta riippuen sallittu tai kielletty, kuten mikä tahansa muukin järjestelmän tuloste. Käytännössä ei voida olla varmoja, onko testikohteen vastaamattomuus hiljaisuutta vai onko vaste vain pahasti myöhässä. Siksi usein sovitaan jostakin ajasta, jonka jälkeen vasteen puute tulkitaan hiljaisuudeksi. Liian pitkä viive ei yleensä muutenkaan ole toivottua, vaan sekin on usein virhetilanne.

Tilakoneiden teoriaan testauksen näkökulmasta johdattaa artikkeli [6].

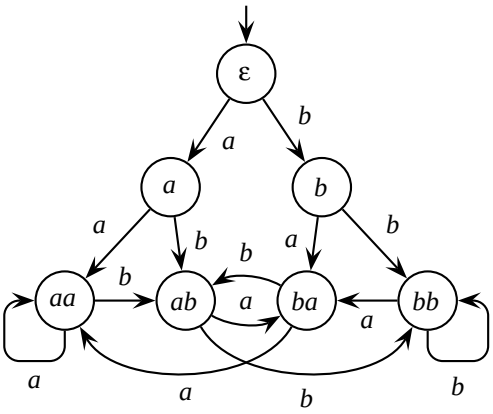
Mallipohjaisessa testauksessa järjestelmän oikea toiminta formalisoidaan tilakonemuotoisella spesifikaatiolla. Tällöin testitapauksia voidaan luoda tämän spesifikaation pohjalta koneellisesti. Testitapaukset voidaan luoda niin, että ne täyttävät jonkin kattavuusvaatimuksen. Esimerkiksi voidaan taata, että luodut testitapaukset ajamalla tulevat kaikki spesifikaation tilasiirtymät käydyksi läpi. Näin testauksesta voidaan tehdä täysin automaattista, kunhan ihminen on ensin tehnyt spesifikaatiotilakoneen.

Spesifikaatiot voivat olla monimutkaisellekin järjestelmälle melko pieniä, etenkin jos ne ovat kovin korkealla abstraktiotasolla. Esimerkiksi monimutkaisenkin protokollan spesifikaatio voisi yksinkertaisimmillaan koostua vain syötteestä ”lähetä viesti *x*” ja sille toivotusta vasteesta ”viesti *x* saapui”. Tällöin edellä kuvattu automaattinen testausmenetelmä ei välttämättä tuota tarpeeksi monta erilaista testitapausta. Epädeterminismin ansiosta rinnakkaisessa järjestelmässä voidaan toki toistaa samoja testejä yhä uudestaan ja toivoa havaittavan uudenlaista käyttäytymistä, mutta tämä ei ole kovinkaan tehokasta.

Samoilutestaus on menetelmä, jossa testausta ei jaeta erikseen yhtä asiaa testaaviin testitapauksiin, vaan testattavat asiat päätetään lennossa pitäen jatkuvasti kirjaa, missä spesifikaation tilassa testiajo kulkee [3]. Seuraavaksi suoritettava tapahtuma valitaan joko satunnaisesti tai käyttäen jotakin heuristiikkaa. Menetelmällä on useita etuja tavalliseen testaukseen verrattuna. Sen avulla muun muassa vältetään järjestelmän turhien ylös- ja alasajosekvenssien toistoa. Lisäksi epädeterminististen järjestelmän tapauksessa voidaan päättää testattava asia sen perusteella, mitä järjestelmä sattuu kullakin kertaa tekemään. Samoilutestauksessa yksittäistä testiajoa voidaan jatkaa tarvittaessa loputtomiin.

4 Historialaajennus

Historialaajennus on tekniikka, jonka tavoitteena on käyttää annettuja testausresursseja mahdollisimman tehokkaasti. Sen kantava ajatus on siinä, että virheet paljastuvat tehokkaammin, kun testataan mahdollisimman monta erilaista *historiaa* eli tilaan johtavien peräkkäisten tapahtumien erilaista yhdistelmää.



Kuva 1: Historialaajennin tapahtumille a ja b, $k = 2$.

Historialaajennuksen idea on hyvin intuitiivinen. Eräs esimerkki, joka ei ole ohjelmistojen testauksen kannalta kovin relevantti, mutta havainnollistaa hyvin historioita, on pankkikortin unohtuneen nelinumeroisen tunnusluvun arvaaminen (kuvitellaan, että yrityksiä olisi käytössä rajattomasti). On selvää, että samaa numeroa ei kannata kokeilla monta kertaa, vaan järkevintä on kokeilla järjestelmällisesti kaikkia eri yhdistelmiä. Tämä on itse asiassa lähellä erilaisten neljän numeron mittaisten historioiden käyttämistä.

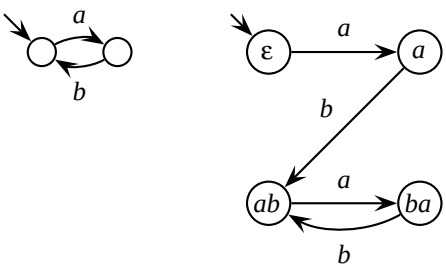
Erilaisia historioita eli yhdistelmiä päästään automaattisesti testaamaan laajentamalla spesifikaatiota sopivalla tavalla. Valitaan jokin pieni kokonaisluku, k, joka on muistettavan historian maksimipituus. Jokainen spesifikaation tila korvataan joukolla tiloja, joista kukin vastaa yhtä (enintään) k:n mittaista korvattuun tilaan johtavaa historiaa.

Spesifikaation laajennus voidaan automatisoida. Alkuperäinen spesifikaatio kytketään rinnan k-historialaajentimen kanssa, ja tuloksena saadaan täsmälleen haluttu laajennettu spesifikaatio.

Kuva 1 esittää k-historialaajenninta, joka on tilakone. Sen jokaisesta tilasta lähtevät kaikki spesifikaation näkyvät tilasiirtymät. Laajentimessa on tila jokaista enintään k:n mittaista tilasiirtymien nimien jonoa eli historiaa kohden. Nämä saadaan siis mekaanisesti rakentamalla kaikki k:n mittaiset ja lyhyemmät yhdistelmät tilasiirtymien nimistä. Tilat, joiden tilasiirtymien jono on k:ta lyhyempi, liittyvät tilakoneen ajon alkuun, jolloin tilakone ei ole vielä ottanut k:ta askelta, joten vastaavaa k:n mittaista historiaa ei ole myöskään olemassa.

Historialaajentimessa tilasta lähtevät tilasiirtymät johtavat aina sellaiseen tilaan, johon liittyvä historia saadaan lisäämällä lähtötilan historiaan suoritettavan tilasiirtymän nimi ja tarvittaessa unohtamalla alkupäästä yksi siirtymä, jotta historia ei kasva pidemmäksi kuin k. Toisin sanoen, jos tilaa vastaava historia eli tilasiirtymien nimien jono on muotoa $a_1 a_2 \cdots a_k$, niin siitä lähtevä tilasiirtymä b vie tilaan, jonka historia on muotoa $a_2 \cdots a_k b$.

Kuva 1 esittää historialaajentimen tapauksessa, jossa tilasiirtymien joukko on



Kuva 2: Vasemmalla yksinkertainen spesifikaatio ja oikealla sen historialaajennettu versio ($k = 2$).

$\{a, b\}$ ja $k = 2$. Kuvassa ε tarkoittaa tyhjää merkkijonoa eli ainoaa mahdollista historiaa tilakoneen käynnistyessä.

Historialaajennin ei rajoita rinnankytkettäessä spesifikaation toimintaa, koska sen jokaisesta tilasta lähtevät kaikki mahdolliset tapahtumat. Niinpä kaikki järjestelmän laillinen toiminta on rinnankytkennän tuloksessa mukana. Toisaalta laajennin ei sisällä muita tapahtumanimiä kuin spesifikaation jo sisältämät, eikä lainkaan näkymättömiä tilasiirtymiä. Siksi se ei salli mitään uutta spesifikaation salliman käyttäytymisen lisäksi.

Useimpiin spesifikaation tiloihin ei voi päätyä kaikilla mahdollisilla historioilla, joten tiloille ei laajennettaessa tyypillisesti synny kopioita kuin murto-osa teoreettisesta maksimimäärästä. Esimerkiksi tilakoneessa, joka koostuu pelkästään peräkkäisten tapahtumien *a* ja *b* yhdessä muodostamasta silmukasta takaisin alkutilaan, kuten kuvassa 2, voi alkutilaan päästä äärettömän monella tapahtumajonolla, nimittäin $\varepsilon, ab, abab, ababab$ ja niin edelleen. Jos $k = 2$, on erilaisia historioita kuvan 1 tiloja vastaavat 7 kappaletta kumpaakin spesifikaation tilaa kohti, joten tilakoneiden karteesisessa tulossa olisi 14 tilaa. Näistä tiloista saavutettavia on vain neljä, alkutilalle historiat ε ja *ab* ja toiselle tilalle historiat *a* ja *ba*.

Käytettäessä historialaajennusta automaattisesti voisi järjestelmä esimerkiksi laajentaa spesifikaatiota kasvavilla *k*:n arvoilla ajaen kullekin arvolle sellaisen joukon testitapauksia, että kaikki laajennetun spesifikaation tilasiirtymät tulevat katetuiksi. Näin voitaisiin jatkaa, kunnes resurssit loppuvat.

Vaikka historialaajennus on yllä esitetty lähinnä tavallisten tilakoneiden näkökulmasta, se toimii lähes sellaisenaan myös syöte/tuloste-jaon kanssa. Historialaajentimessa asetetaan kukin spesifikaation syöte tulosteeksi ja tuloste syötteeksi, jotta rinnankytkentä voi synkronoida syöte/tuloste-pareja. Erikoistapahtuma δ eli hiljaisuus aiheuttaa hieman ongelmia. Jos se olisi vain yksi ylimääräinen näkyvä tapahtuma muiden joukossa, niin järjestelmää aina välillä kuunneltaessa tule-

vat hiljaisuudet sotkisivat historioita osittain mielivaltaisella tavalla (riippuen siitä, koska järjestelmää satutaan kuuntelemaan). Asia käy selkeästi ilmi esimerkissä, joka esitetään kohdassa 5.4. Tämän takia hiljaisuutta ei tässä lasketa tulosteeksi historioiden kannalta (hiljaisuus on silti hyvä keino saada kiinni sellaisia virhetilanteita, joissa järjestelmältä edellytetään jotain reaktiota, mutta se ei sitä anna).

Historialaajennin on esitetty erillisenä tilakoneena lähinnä ymmärrettävyyssyistä. Kun historialaajennusta toteutetaan käytännössä, voidaan laajennus toteuttaa automaattisesti myös suoraan spesifikaatiota muokaten. Näin vältytään käsittelemästä eksplisiittistä historialaajennintilakonetta, joka olisi turhana välivaiheena mahdollisesti paljonkin lopullista laajennettua spesifikaatiota suurempi.

5 Kokeellista tehokkuusvertailua

Intuitiivisesti tuntuu luonnolliselta, että mahdollisimman monen erilaisen historian tutkiminen paljastaa virheitä tehokkaasti. Erilaisten testausmenetelmien vertailu on hankalaa [2], mutta tehokkuutta on mahdollista tutkia puolueettomilla kokeilla jossain määrin.

5.1 Koejärjestelyt

Koejärjestelyissä tutkitaan tyypillisesti jotakin järjestelmää, johon on tahallaan istutettu virhe. Järjestelmästä rakennetaan myös tilakone, joka spesifioi sen oikean toiminnan jollakin abstraktiotasolla. Järjestelmää testataan toisaalta alkuperäisellä spesifikaatiolla ja toisaalta erilaisilla k:n arvoilla historialaajennetuilla spesifikaatioilla ja verrataan tuloksia.

Tällaisella testijärjestelyllä voitaisiin myös tutkia esimerkiksi sitä, kannattaako spesifikaatiot minimoida, ennen kuin historialaajennusta sovelletaan niihin. Alkuperäisessä spesifikaatiossa oleva redundanssi on itse asiassa osittaista historialaajennusta ja sikäli turhaa, koska historialaajennus kasvattaa spesifikaatiota hallitusti.

Tällaiset kokeet voitaisiin suorittaa käyttäen kyseisen järjestelmän oikeaa toteutusta, mutta jos järjestelmä on riittävän pieni, myös toteutus voidaan mallintaa (spesifikaatiota yksityiskohtaisempana) tilakoneena. Toteutukseen voidaan silloin kätkeä virheitä, jolloin on mahdollista laskea suoraan tilakoneiden rinnankytkennästä virheiden löytämisen todennäköisyys ja testiajon hinta joko ratkaisemalla (valtava) yhtälöryhmä tai käyttäen tätä tarkoitusta varten kehitettyä algoritmia [4].

Testit voitaisiin suorittaa yrittämällä kattaa annetun spesifikaation kaikki kaaret tai tilat, mutta näissä kokeissa on käytetty samoilutestausta ahneen satunnaisuuden heuristiikalla. Ahneen satunnaisuuden heuristiikka tarkoittaa, että testaaja lähettää vaihtoehtoisista syötteistä aina sen, joka vastaa spesifikaation kaarta, jota on tähän mennessä suoritettu vähiten. Tasatilanteessa syöte valitaan satunnaisesti.

Samoilutestaus jatkaa spesifikaation läpikäymistä kunnes virhe löytyy. Niinpä kustakin spesifikaation ja järjestelmän muodostamasta kokonaisuudesta voidaan laskea yksi luku, joka kertoo, kuinka monta tilasiirtymää joudutaan keskimäärin tekemään ennen kuin virhe löytyy.

Paitsi ahneen satunnaisuuden heuristiikalla, useimmat testit ajettiin myös täysin satunnaisella heuristiikalla, joka valitsee kussakin tilassa satunnaisesti jonkin mahdollisista syötteistä. Täysi satunnaisuus on usein yllättävän hyvä testaustekniikka, sillä se tulee käytännössä lopulta

kokeilleeksi kaikkea mahdollista, kun taas ahne satunnaisuus voi päätyä testaamaan asioita aina samassa järjestyksessä, ja näin virhe voi jäädä kokonaan havaitsematta.

5.2 Vuorottelevan bitin protokolla

Vuorottelevan bitin protokolla [1] on eräs yksinkertainen protokolla, jolla voidaan välittää tietoa epävarmojen tiedonsiirtokanavien yli (viesti voi hukkua kanavassa). Protokollassa jokaiseen lähetettävään viestiin liitetään ylimääräinen tarkistusbitti, jonka arvo vuorottelee peräkkäisten viestien välillä. Lähettävä ja vastaanottava pää pitävät kirjaa vuorottelevan bitin arvosta, ja vastaanottaja hyväksyy vain sellaiset viestit, joiden bitin arvo on odotettu. Perille menneet viestit kuitataan, ja kuittaukseen liitetään myös vuorotteleva bitti. Kun oikealla bitillä varustettu kuittaus tulee perille lähettäjälle, se voi siirtyä lähettämään seuraavaa viestiä; muussa tapauksessa se lähettelee edellistä viestiä uudestaan aina silloin tällöin.

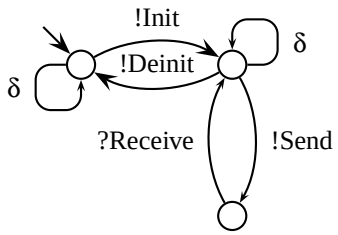
Äkkiseltään voi tuntua, että protokollaa voitaisiin yksinkertaistaa poistamalla kuittausviesteistä vuorotteleva bitti. Tämä kuitenkin johtaa mielenkiintoiseen virhetoimintoon, sillä protokolla voi tässä muodossaan hukata viestejä, mutta ei kuitenkaan ensimmäistä viestiä.

Protokollissa on yleensä jokin yhteyden luomisvaihe ja mahdollisesti yhteyden katkaisuvaihe. Virhe saattaisi esiintyä myös niissä, esimerkiksi voitaisiin ajatella, että jossakin protokollassa yhteyden katkaisussa jäisi jotakin nollaamatta tai poistamatta ja virhe voitaisiin löytää vain katkaisemalla välillä yhteys ja muodostamalla se uudestaan. Niinpä kattavaa testausta etsittäessä tulee tällaisetkin tapaukset protokollan toiminnasta testata. Kuvan 3 tilakone esittää testattavan protokollan spesifikaation hyvin abstraktilla tasolla. Meitä ei esimerkiksi kiinnosta viestien sisältö.

Järjestelmä toteutettiin varsin abstraktina mallina ja rohkein oletuksin. Rikkinäisen vuorottelevan bitin protokollan mukaisesti järjestelmä lähettää ensimmäisen viestin aina oikein, ja toisella sekä sitä seuraavilla viesteillä on tietty vakiotodennäköisyys kadota. Yksinkertaisuuden säilyttämiseksi järjestelmä toteutettiin toimimaan täsmälleen näin, eli edes toteutuksen tasolla ei ollut olemassa vuorottelevia bittejä. Lisäksi oletimme, että protokollassa yhteys täytyisi luoda ja katkaista erikseen, vaikka itse vuorottelevan bitin protokollassa tällaisia vaiheita ei ole. Saatu toteutus kytkettiin rinnan samoilutestauksen toimintaa erilaisissa tapauksissa kuvaavien tilakoneiden kanssa. Tutkitut tapaukset ovat *täysi satunnaisuus* ja *historialaajennus* k:n arvoilla 0 ja 1. Historialaajennuksessa käytettiin *ahneen satunnaisuuden* heuristiikkaa. Oikeastaan tapausta, jossa k:n arvo on 0 voisi kutsua pelkäksi ahneeksi satunnaisuudeksi, sillä tällöin spesifikaatio on alkuperäinen.

Samoilutestauksessa täyden satunnaisuuden heuristiikalla historialaajennuksella ei itse asiassa ole mitään merkitystä, koska historialaajennetussa spesifikaatiossa kuvataan täsmälleen sama toiminta kuin alkuperäisessä, joten satunnaisuus tekee samat testit samoilla todennäköisyyksillä molemmissa tapauksissa. Tämän takia kaikissa tämän artikkelin testeissä historialaajennuksen yhteydessä on käytetty ahnetta satunnaisuutta (ja tapausta $k = 0$ voidaan ajatella pelkäksi ahneeksi satunnaisuudeksi ja pitää näin pääasiallisena vertailukohtana historialaajennuksen tehokkuutta ajatellen).

Yhteyden muodostamista ja katkaisemista sekä viestin lähettämistä (eli testauksen syötteitä) kohdeltiin täysin tasa-



Kuva 3: Tiedonsiirtoprotokollan abstrakti spesifikaatio.

arvoisina toisiinsa nähden. Toisin sanoen jokainen niistä oli ennen heuristiikkojen vaikutusta yhtä todennäköinen ja samanhintainen (1 yksikkö).

Oheinen taulukko esittää keskimääräisiä hintoja virheen löytymiselle eräillä virhetodennäköisyyksillä. Taulukossa p on virheen paljastumisen todennäköisyys kullakin lähetetyllä viestillä (toisesta alkaen).

p	$k=1$	$k=0$	satunn.
1	6,5	11	10
0,2	30,5	59	34
0,01	600,5	1 199	604
10^{-4}	59 999,5	119 999	60 004

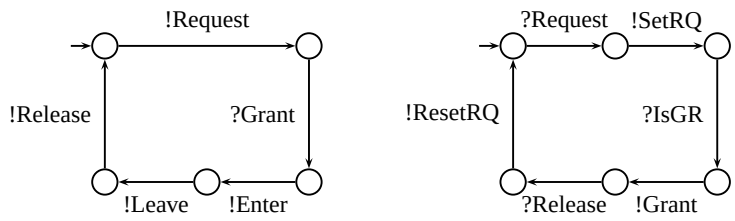
Tulokset ovat sikäli tarkkoja, että ne ovat tilakoneiden rinnankytkennöistä laskettuja odotusarvoja eivätkä kokeellisia keskiarvoja. Toisaalta järjestelmää on abstrahoitu niin paljon, että lukujen ja todellisuuden vastaavuudesta ei juuri voi puhua. Niiden vertaaminen toisiinsa on silti mielenkiintoista.

Tuloksista on havaittavissa, että historialaajennus on tämän virheen etsimisessä vain hitusen parempi kuin puhdas satunnaisuus mutta noin kaksi kertaa niin tehokas kuin pelkkä ahne satunnaisuus.

Ahneen satunnaisuuden epävarmuus erilaisten historioiden kokeilemisessa näkyy tässä järjestelmässä. Jos se nimittäin sattuu alussa pakollisen yhteyden luomisen jälkeen ensimmäiseksi kokeilemaan viestin lähettämistä, sen on seuraavaksi pakko katkaista yhteys toisen (ja mahdollisesti virheen paljastavan) viestin lähettämisen sijaan, jolloin tämä kaikki on haaskattua työtä. Vain kokeilemalla ensin sattumalta yhteyden katkaisua ilman yhdenkään viestin lähettämistä se voi päätyä kokeilemaan kahden viestin lähettämistä peräkkäin. Jos virheen löytäminen vaatisi kolmen peräkkäisen viestin lähettämistä, se ei löytäisi virhettä ikinä!

Tavallinen satunnaisuus sen sijaan voi kyllä kokeilla kahden viestin lähettämistä perätysten, mutta toisaalta se voi myös juuttua loputtomasti katkomaan ja muodostamaan yhteyttä lähettämättä koskaan ensimmäistäkään varsinaista viestiä. Historialaajennus taas tulee jo k:n arvolla 1 kokeilleeksi tasaisesti monenlaisia yhdistelmiä.

Tämän kokeen perusteella historialaajennus on hyvä yhdistelmä ahneen satunnaisuuden ja puhtaan satunnaisuuden eduista. Historialaajennus kokeilee järjestelmällisesti kaikenlaisia yhdistelmiä, jolloin säilytetään puhtaan satunnaisuuden hyvä puoli, eli se, että järjestelmään ei jää sellaisia suorituksia, joita ei koskaan



Kuva 4: Vasemmalla testaajan spesifikaatio yhden asiakkaan osalta. Oikealla asiakkaan toteutus, jossa SetRQ ja ResetRQ muuttavat pyyntömuuttujan arvoa ja IsGR tarkistaa, onko lupa saatu.

edes teoriassa voitaisi suorittaa, kunhan k:n arvo on tarpeeksi suuri tai sitä kasvatetaan testiajon jatkuessa. Liian pienellä k:n arvolla nimittäin saman toiminnon jatkuvaa toistoa vaativat virheet eivät jää kiinni. Toisaalta historialaajennus hyötyy ahneen satunnaisuuden heuristiikan yleensä ottaen paremmasta tehokkuudesta.

5.3 Kaksivaihekättely

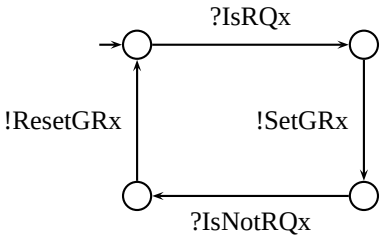
Niin sanottu kaksivaihekättely on järjestelmä, jossa palvelin säätelee asiakkaiden pääsyä johonkin kriittiseen resurssiin. Kunkin asiakkaan i ja palvelimen kommunikointi tapahtuu kahdella jaetulla binäärisellä muuttujalla, p_i (pyyntö) ja l_i (lupa). Kun asiakas i haluaa resurssin käyttöönsä, se asettaa muuttujan p_i todeksi ja odottaa, että palvelin antaa luvan käyttää resurssia asettamalla vastaavasti muuttujan l_i todeksi. Kun asiakas on valmis, se vapauttaa resurssin asettamalla p_i:n epätodeksi ja odottaa, kunnes palvelin huomaa vapautuksen, jolloin palvelin asettaa muuttujan l_i taas epätodeksi. Jos useampi asiakas pyytää lupaa yhtä aikaa, palvelin päättää satunnaisesti, mikä niistä saa luvan käyttää resurssia.

Järjestelmän hahmottamista auttavat tilakoneet esitetään kuvissa 4 ja 5.

Järjestelmää toteuttaessa on helppo tehdä pieni virhe ja päästää asiakas jatkamaan toimintaansa jo siinä vaiheessa, kun se on lakannut käyttämästä resurssia ja vapauttanut sen asettamalla p_i:n epätodeksi. Näin toteutettuna järjestelmä toimii yllättäen väärin. Jos sama asiakas pyytää resurssia pian uudestaan, se voi ehtiä tulkita edelliseltä kierrokselta todeksi jääneen l_i:n luvaksi käydä uudestaan resurssin kimppuun, mutta samalla palvelin onkin antamassa lupaa jo seuraavalle asiakkaalle ja näin molemmat pääsevät yhtä aikaa käsittelemään resurssia, mistä usein seuraa kaaos.

Toteutettaessa tilakoneena järjestelmää, joka käyttää ahnetta satunnaisuutta, täytyy testaajaa kuvaavassa tilakoneessa pitää kirjaa siitä, montako kertaa mikäkin spesifikaation siirtymä on suoritettu (käytännössä täytyy muistaa vain, mitä kunkin tilan siirtymistä on suoritettu enemmän kuin muita). Koska testaustilakoneen koko kasvaa eksponentiaalisesti suhteessa spesifikaation kokoon, on ahnetta satunnaisuutta vaikea mallintaa tilakoneilla.

Niinpä tämä järjestelmä toteutettiin ohjelmana, jolle tilakonetta spesifikaationa käyttävä testiympäristö antaa syötteitä ja kuuntelee sen vasteita. Niinpä tulokset ovat kokeellisten toistojen keskiarvoja ei-



Kuva 5: Palvelimen toteutus asiakkaan *x* osalta. SetGRx ja ResetGRx muuttavat kyseisen asiakkaan lupamuuttujan arvoa ja IsRQx sekä IsNotRQx tarkistavat, haluaako asiakas luvan sekä onko asiakas vapauttanut resurssin. Palvelimessa on jokaista asiakasta varten vastaava silmukka alkutilasta.

vätkä tarkasti laskettuja odotusarvoja, kuten edellisessä esimerkissä.

testaustapa	hinta
Satunnainen	21 860
Historialaajennettu, $k = 0$	22 053
Historialaajennettu, $k = 1$	21 686
Historialaajennettu, $k = 2$	22 290

Seuraava taulukko näyttää muutamia edellä kuvatun virheen etsimiseen liittyviä tuloksia. Testaukseen käytettiin jälleen samoilutestausta, jossa järjestelmään lähetettävät syötteet valittiin joko täysin satunnaisesti tai ahneen satunnaisesti. Ahneen satunnaisuuden tapauksessa käytettiin sekä alkuperäistä spesifikaatiota ($k = 0$) että kahdella eri positiivisella k:n arvolla historialaajennettua spesifikaatiota. Taulukon lukuarvot kertovat keskimääräisen tapahtumien määrään ennen kuin virhe havaittiin. Taulukon tulokset koskevat kahden asiakkaan järjestelmää. Testejä tehtiin eri asiakasmäärillä aina seitsemään asti ja historialaajennuksen k-parametrin arvoilla kuuteen asti, mutta tulokset ovat kaikilla arvoilla hyvin samankaltaiset. Virheen löytäminen hidastuu kaikilla testaustavoilla yhtä lailla asiakasmäärän kasvaessa ja on suunnilleen yhtä tehokasta testaustavasta riippumatta. Luvut on saatu 10 000 toistolla.

Erot tuloksissa ovat melko pieniä ja niitä voidaan pitää lähinnä satunnaisuuden tuottamina. Vaikka mitään tilastollista analyysiä aineistolle ei tehtykään, niin tähän viittaa myös se, että testejä toistamalla eri menetelmien paremmuusjärjestys vaihteli satunnaisen oloisesti.

Erojen puutetta selittää se, että testaajalla ei ole tässä järjestelmässä kovinkaan paljoa valinnanvaraa. Virhe ilmenee lähes testaajasta riippumatta, kunhan virheelle välttämättömät kaksi ehtoa on täytetty (ainakin kaksi asiakasta on pyytänyt resurssia ja ainakin yksi asiakas on tehnyt sen vähintään kahdesti).

Ainoa tapa testata järjestelmää epäoptimaalisesti on siis testata pelkästään tapausta, jossa yksi asiakas pyytää resurssia jatkuvasti. Puhdas satunnaisuus on käytetyistä menetelmistä ainoa, joka toimii näin joskus. Näin aiheutunut virheen hitaampi löytyminen katoaa tuloksista, koska virheen paljastumiseksi täytyy tehdä tyypil-

lisesti tuhansia pyyntöjä, mutta todennäköisyys sille, että aito satunnaisuus tekee kaikki nämä tuhannet pyynnöt yhdellä ja samalla asiakkaalla, on häviävän pieni.

5.4 Tunnusluvun arvaaminen

Jo aiemmin tässä tekstissä esimerkkinä käytetty tunnusluvun arvaaminen on edellisistä poiketen keinotekoisesti valittu esimerkki, jossa historialaajennuksen etujen pitäisi tulla hyvin esiin. Esimerkki ei liity suoranaisesti testaukseen, mutta sen kaltaisia tapauksia on testauksesta löydettävissä. Tällainen tilanne syntyy esimerkiksi, jos testattava järjestelmä on sellainen, että useimmiten on valittavissa melkein mikä tahansa syöte, mutta virheen paljastuminen vaatii jonkin tietyn pitkän syötesarjan. Arvattava tunnusluku vastaa siis tässä sitä, että testaaja arvaa tämän syötesarjan, jolla virhe paljastuu.

Yksinkertaistetaan pankkikortin neljän numeron mittaisen tunnusluvun arvaamisen spesifikaatio äärimmilleen, jolloin siinä on vain yksi tila, jossa voidaan syöttää mikä tahansa numeroista. ”Virhe” tapahtuu, kun numero saadaan oikein, jolloin järjestelmä lakkaa kuuntelemasta syötettä. ”Testattava” järjestelmä siis tarkastaa neljän numeron välein, oliko viimeksi syötetty nelikko oikea tunnusluku, mutta tämä ei yksinkertaistetusta spesifikaatiosta käy ilmi. Mitä nopeammin luku arvataan eli virhe löydetään, sitä paremmin testaus on sujunut. Tässä versiossa ei siis tarvitse painaa ok-nappulaa, eikä spesifikaatiosta muutenkaan tiedetä, kuinka pitkää lukua etsitään (tämä vastannee testauksessa vastaan tulevia tilanteita paremmin kuin aito pankkiautomaatti). Kun oikea luku arvataan, testaus pysähtyy ja lasketaan, montako kokeilua tarvittiin.

Historialaajennus ei vaikuta olevan kovin paljon puhdasta satunnaisuutta tehokkaampi testauskeino tällaisessakaan tehtävässä. Pelkkä ahne satunnaisuus on selvästi huono ratkaisu, sillä se ei tule kokeilleeksi kaikkia mahdollisuuksia. Esimerkiksi yhdistelmä 1111 on sille mahdoton arvattava, koska viimeistään kahden ykkösen jälkeen se on kokeillut muita numeroita vähemmän kuin ykköstä, joten se ei voi jatkaa yhdistelmää ykkösellä.

Puhtaan satunnaisuuden ja historialaajennuksen eroja vertaillaan seuraavassa taulukossa. Lisäksi taulukossa on optimaalinen tulos, joka saavutettaisiin testaamalla järjestelmällisesti eri tunnusluvut. Puhtaan satunnaisuuden ja järjestelmällisyyden tulokset on saatu laskemalla ja historialaajennuksen tulokset simulaatiolla kokeilemalla kussakin kohdassa 1 000 satunnaisen yhdistelmän arvaamista. Sarakkeet kuvaavat tunnusluvun pituutta, *rnd* on puhdas satunnaisuus, *optimi* on tulos, joka saavutettaisiin kokeilemalla eri tunnuslukuja järjestelmällisesti ja eri k:n arvot ovat historialaajennettuja versioita (mukaan lukien pelkkä ahne satunnaisuus).

	1	2	3	4
rnd	10	100	1000	10 000
optimi	5,5	51	501	5 001
$k = 0$	8,7	185	∞	∞
$k = 1$	9,4	87	1 063	∞
$k = 2$	9,8	92	911	∞
$k = 3$	9,9	98	931	9 209
$k = 4$	9,9	100	1 001	9 173

Toisessa taulukossa ovat samat tulokset, mutta keskiarvojen sijaan luvut ovat tulosten mediaaneja. Näin on saatu tuloksia myös pienille k:n arvoille, sillä äärettömyyksiä aiheuttavia tunnuslukuja on vain pieni murto-osa kaikista, joten ne eivät vaikuta mediaaniin.

	1	2	3	4
$k = 0$	6	70	777	9 988
$k = 1$	7	62	677	7 089
$k = 2$	7	66	617	7 046
$k = 3$	7	69	669	6 445
$k = 4$	7	71	686	6 728

Tuloksista havaitaan, että vaikka historialaajennus vaikuttaa olevan lähes aina hieman tehokkaampaa kuin satunnaisuus, ero on melko pieni. Lyhyemmillä luvuilla on nähtävissä, että *k*-arvon liiallinen kasvattaminen huonontaa tuloksia. Ilmiö selittyy sillä, että tunnusluvun pituutta isommilla *k*:n arvoilla yhteen täysimittaiseen historiaan sisältyy enemmän kuin yhden tunnusluvun verran numeroita, joten samaa lukua voidaan kokeilla useita kertoja ennen kuin kaikkia on kokeiltu ensimmäisen kerran. Esimerkiksi yhden mittaisia lukuja arvattaessa *k*:n arvolla 2 ovat historiat 51 ja 35 erilaisia, mutta niiden kokeileminen tulee kokeilleeksi lukua 5 kahdesti.

Järjestelmällinen tunnuslukujen kokeileminen on koetulosten perusteella tehokkaampaa kuin historialaajennus, vaikka edellä mainittiin, että historialaajennus vastaa likimain järjestelmällistä kokeilua. Tämä johtuu siitä, että historialaajennus ei tässä tapauksessa erota uuden numerosarjan alkua, joten se tulee kokeilleeksi samoja yhdistelmiä uudestaan, ennen kuin se kokeilee kaikkia ensimmäisen kerran. Esimerkiksi kahden mittaisilla tunnusluvuilla ja *k*:n arvolla 2 se voisi kokeilla aluksi 36 ja sitten 71, minkä jälkeen se olisi jo mielestään kattanut kahden mittaisen historian 67 ja tulisi kokeilleeksi sitä aikaisintaan kattaessaan historioita toista kertaa, eikä silloinkaan välttämättä.

Edellä kuvattu historialaajennuksen tässä tehtävässä esiintyvä ongelma on kenties yleisempikin ilmiö. Jos kaksi toteutuksen kannalta jollakin tavalla hyvin erilaista tilaa kuvautuvat spesifikaatioabstraktiossa samaksi tilaksi, saattaa historialaajennus jättää jotain oleellista testaamatta.

Lisäksi huomattiin, että ahne satunnaisuus ei selviydy kaikista tunnusluvuista. Tämä koskee myös historialaajennettuja versioita (jotka siis käyttävät ahnetta satunnaisuutta laajennuksen jälkeen), jos *k* on liian pieni. Esimerkiksi neljän mittaista tunnuslukua 1111 ei voida koskaan arvata arvolla $k = 1$, koska tietyn historian jälkeen voidaan sama valinta tehdä enintään kahdesti, eli historian 1 jälkeen ei voida valita kolmesti peräkkäin numeroa 1. Ongelmasta päästään eroon, kun *k*:n arvo on tarpeeksi suuri. Tämä on kuitenkin ongelma, sillä järjestelmästä ei välttämättä näe ulospäin, mikä *k*:n arvon pitäisi olla. Sopivan *k*-arvon valinta tai jonkinlainen adaptiivinen erilaisten *k*-arvojen käyttö saman testiajon aikana voisikin olla eräs mahdollinen jatkotutkimuksen kohde.

Tämän järjestelmän yhteydessä on myös helppo havainnollistaa, mitä tapahtuisi, jos hiljaisuus tulkittaisiin samanlaiseksi tulosteeksi kuin kaikki muutkin. Koska esimerkkijärjestelmämme on hiljaa aina numeron syöttämisen jälkeen, voivat hiljaisuudet sotkea historian pahasti. Esimerkiksi luvun 1234 syöttämisen jälkeen historia voisi olla 1234, mutta yhtä hyvin esimerkiksi 3δ4δ, jos järjestelmää olisi satuttu kuuntelemaan välillä. Tällöin mikään ei estäisi järjestelmää kokeilemasta lukua 1234 uudestaan, kunhan mahdolliset kuuntelut suoritettaisiin eri kohdissa.

Järjestelmästä voidaan tehdä myös sellainen versio, jossa spesifikaatiossa tiedetään, monenko mittaista tunnuslukua ollaan etsimässä. Käytännössä tämä toteutettiin niin, että käyttäjän tulee painaa hyväksymisnappia syötteen jälkeen, jolloin järjestelmä kertoo, oliko arvaus oikein vai väärin.

Seuraavassa taulukossa on esitetty tulokset tällaiselle järjestelmälle. Luvut ovat keskimääräisiä arvausten määriä, ”rnd” on täysi satunnaisuus, ”optimi” on järjestelmällinen kokeilu, ”$k = 0$” on ahne satunnaisuus ja loput ovat historialaajennuksen eri tapauksia. Sarakkeissa on tunnusluvun pituus.

	1	2	3	4
rnd	10	100	1000	10 000
optimi	5,5	51	501	5 001
$k = 0$	5,5	94	1 011	9 983
$k = 1$	5,9	50	954	9 980
$k = 2$	6,0	50	501	9 356
$k = 3$	9,3	51	504	5 001
$k = 4$	9,2	89	499	5 011

Kuten oli odotettavissa, hyväksymisnapin painaminen poistaa aiemmassa tapauksessa esiintyneet ongelmat uuden numerosarjan erottamisesta ja ahneen satunnaisuuden kyvyttömyyden arvata kaikkia tunnuslukuja. Näin ollen historialaajennuksen tulokset ovat hyvin lähellä optimia, kunhan k:n arvo ei ole liian pieni tai liian suuri. Tämä on oikeastaan sen ansiota, että spesifikaatio on hyvin lähellä itse järjestelmää. Spesifikaation erottaa toteutuksesta ainoastaan se, että toteutus tietää oikean tunnusluvun, mutta spesifikaatio ei.

6 Yhteenveto

Tässä artikkelissa esiteltiin historialaajennus, joka on mallipohjaisen testauksen tehostamiseen tarkoitettu automaattinen menetelmä. Sen idea on intuitiivinen ja vastaa jossakin mielessä sitä tapaa, jolla ihminen järjestelmällisesti testaisi erilaisia tapahtumaketjuja. Tehokkuustestien merkitystä arvioitaessa kannattaa muistaa, että historialaajennusta ei ole kehitetty tehostamaan testejä, vaan sen tarkoitus on helpottaa kattavuuksien hallintaa generoimalla spesifikaatiosta alkuperäistä laajempia versioita.

Historialaajennukseen liittyvän teorian lisäksi esiteltiin koetuloksia, joita on saatu laskemalla ja kokeilemalla, kuinka nopeasti eräisiin järjestelmiin tahallaan istutettu virhe löytyy toisaalta historialaajennuksen avulla ja toisaalta joillakin tavallisilla mallipohjaisessa testauksessa käytettävillä menetelmillä.

Historialaajennuksen käyttäminen ei useimmissa kokeilluissa tapauksissa tehostanut testausta merkittävästi, mutta siitä ei ollut merkittävää haittaakaan. Etenkin järjestelmissä, joissa on vain vähän erilaisia tapahtumia tai testaajalla on muuten vain vähän mahdollisuuksia vaikuttaa siihen, mitä testataan, historialaajennuksen tulokset ovat samanlaisia kuin esimerkiksi puhtaan satunnaisuuden tulokset. Eräissä osittain tahallaan historialaajennukselle hyödyllisiksi tunnetuissa tapauksissa virheen löytämiseen kuluva aika noin puolittui (tai vastaavasti todennäköisyys löytää jokin epädeterministisyyden kätkemä virhe annetussa ajassa kaksinkertaistui) puhtaaseen satunnaisuuteen verrattuna.

Suoritetut testit koskivat vain muutamaa yksinkertaista järjestelmää. Niiden avulla kuitenkin saatiin esiin useita historialaajennukseen liittyviä ilmiöitä, joiden voidaan arvella esiintyvän yleisemminkin erilaisia järjestelmiä testattaessa.

Kiitokset

Työ tehtiin Tampereen teknillisen yliopiston ohjelmistotekniikan laitoksella TEKESin, Conformiq Software Oy Ltd:n ja Nokian tutkimuskeskuksen rahoittamassa SASOKE-projektissa. Kiitos Antti Valmarille ja Tiiti Kellomäelle kommenteista.

Viitteet

[1] K. Bartlett, R. Scantlebury & P. Wilkinson: A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM 12 (5)*, 1969, ss. 260–261.

[2] R. Hamlet: Theoretical comparison of testing methods. *Proceedings of the ACM SIGSOFT '89 third symposium on Software testing, analysis and verification*, ss. 28–37, 1989.

[3] J. Helovuo & S. Leppänen: Exploration testing. *Proc. ICACSD 2001, 2nd IEEE International Conference on Application of Concurrency to System Design*, IEEE Computer Society 2001, ss. 201–210.

[4] T. Kellomäki & A. Valmari: A method for analysing the performance of testing techniques for concurrent systems. *Proc. Fifth International Conference on Application of Concurrency to System Design (ACSD 2005)*, ss. 154 - 163.

[5] J. Tretmans: *A Formal Approach to Conformance Testing*. Väitöskirja, Twenten yliopisto, joulukuu 1992.

[6] J. Tretmans: Test generation with inputs, outputs and repetitive quiescence. *Software — Concepts and Tools*, Vol 17(3), Springer-Verlag 1996, ss. 103–120.

[7] A. Valmari: Esimerkki modernista sovellushakuisesta teoriasta: vuorovaikuttavat tilakoneet. *Proc. Tietojenkäsittelytieteen päivät 2004*. Joensuun yliopistopaino, Joensuu, 2004.